UNITED STATES PATENT APPLICATION

FOR

METHOD AND APPARATUS FOR LOADING A TRUSTABLE OPERATING SYSTEM

INVENTORS:

Michael A Kozuch
James A Sutton II
David Grawrock

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

FIELD OF THE INVENTION

[0001] This invention relates to microprocessors. In particular, the invention relates to processor security.

BACKGROUND

[0002] Advances in microprocessor and communication technologies have opened up many opportunities for applications that go beyond the traditional ways of doing business. Electronic commerce and business-to-business transactions are now becoming popular, reaching the global markets at a fast rate. Unfortunately, while modern microprocessor systems provide users convenient and efficient methods of doing business, communicating and transacting, they are also vulnerable to unscrupulous attacks. Examples of these attacks include virus, intrusion, security breach, and tampering, to name a few. Computer security, therefore, is becoming more and more important to protect the integrity of the computer systems and increase the trust of users.

[0003] In the context of operating systems, computer security is determined initially by establishing that you are loading (or have loaded) a trustable operating system. A trustable operating system is where the user or a third party may later inspect the system and determine whether a given operating system was loaded, and if so, whether or not the system was loaded into a secure environment.

[0004] However, when booting a normal operating system it is necessary to boot a wide variety of code components. Even if you could choose what code component should be loaded, the operating system contains such an extremely large amount of code that it is difficult to establish the operating system's specific identity and whether you should choose to trust it, i.e. whether it was loaded into a secure environment.

[0005] In a multi-processor environment, it may be particularly difficult to determine whether the operating system can be trusted. This is because each of the central processing units (CPUs), or sometimes even a system device, can execute a code stream that can potentially alter and compromise the integrity of the code that was loaded. Consequently, at least at the operating system level, it is often necessary to assume that the operating system is trustworthy. Such assumptions may prove to be false and can lead to catastrophic failures in computer security.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

FIG. 1 is a diagram illustrating a generalized overview of the organization of typical operating system components and corresponding privilege levels;

FIG. 2 is a block diagram illustrating one generalized embodiment of a computer system incorporating the invention, and in which certain aspects of the invention may be practiced;

FIG. 3 is a flow diagram illustrating certain aspects of a method to be performed by a computing device executing one embodiment of the illustrated invention shown in FIG. 2;

FIG. 4 is a flow diagram illustrating certain other aspects of a method to be performed by a computing device executing one embodiment of the illustrated invention shown in FIG. 2;

FIG. 5 is a flow diagram illustrating certain aspects of a method to be performed by a computing device executing another embodiment of the illustrated invention shown in FIG. 2; and

FIG. 6 is a block diagram illustrating one generalized embodiment of a computer system in which certain aspects of the invention illustrated in FIGS. 2-5 may be practiced.

DETAILED DESCRIPTION

[0007] In the following description various aspects of the present invention, a method and apparatus for loading a trustable operating system, will be described. Specific details will be set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to those skilled in the art that the present invention may be practiced with only some or all of the described aspects of the present invention, and with or without some or all of the specific details. In some instances, well-known features may be omitted or simplified in order not to obscure the present invention.

[0008] Parts of the description will be presented using terminology commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art, including terms of operations performed by a computer system and their operands, such as transmitting, receiving, retrieving, determining, generating, recording, storing, and the like. As well understood by those skilled in the art, these operands take the form of electrical, magnetic, or optical signals, and the operations involve storing, transferring, combining, and otherwise manipulating the signals through electrical, magnetic or optical components of a system. The term system includes general purpose as well as special purpose arrangements of these components that are standalone, adjunct or embedded.

[0009] Various operations will be described as multiple discrete steps performed in turn in a manner that is most helpful in understanding the present invention. However, the order of description should not be construed as to imply that these operations are necessarily performed in the order they are presented, or even order dependent. Lastly, repeated usage of the phrase "in one embodiment" does not necessarily refer to the same embodiment, although it may.

[0010] One principle for providing security in a computer system or platform is the concept of enforcing privilege levels. Privilege levels restrict which system resources (e.g. privileged instructions, memory, input/output devices and the like) that a particular software component can access. FIG. 1 is a diagram illustrating a generalized overview of the organization of typical operating system components and corresponding privilege levels. In a system without virtual-machine (VM) technology 100, the operating system 120 includes a small resident program component called a privileged software nucleus 125 that operates with the highest privilege level 170, i.e. the privileged software nucleus 125 can execute both privileged and non-privileged instructions and access memory and I/O devices. Another type of system component, the device drivers 130, also execute with a high privilege level 170, particularly if the system supports direct memory access (DMA) transactions, in which a device driver 130 may write the contents of its device directly to memory without involving a processor (e.g. without using the privileged software nucleus 125 to access memory). Still other types of system components, such as the applications 140, operate with a lower privilege level 180 and are only able to execute non-privileged or lesser-privileged instructions or to make supervisory calls (SVCs) to the privileged software nucleus 125 in the operating system 120 to execute the privileged instructions or, more generally, to access privileged system resources on behalf of the application 140.

[0011] In a system with VM technology 110, another type of system component executes with the highest privilege: the virtual-machine monitor (VMM) 150. In a VM system 110, the operating system 120 actually executes with less privilege than the VMM 150. In some VMM implementations, the VMM 150 may be broken into a VMM core component 150 and one or more VMM extensions 160 that execute with less privilege than the VMM core component 150 but more than the operating system 120. In this way,

the VMM core component 150 maintains its integrity in the presence of faulty VMM extensions 160.

[0012] FIG. 2 is a block diagram illustrating one generalized embodiment of a computer system 200 incorporating the invention, and in which certain aspects of the invention may be practiced. It should be understood that the distinction between the various components of computer system 200 is a logical distinction only; in practice, any of the components may be integrated into the same silicon die, divided into multiple die, or a combination of both, without departing from the scope of the invention. In the illustrated computer system 200, either the central processing units (CPU) 210/220/230 or the devices 240/245/250 have the necessary high privilege levels 170 that enable them to initiate transactions in memory 270. The memory controller 260 is responsible for forwarding the memory transaction from memory 270 to the appropriate destination.

[0013] The computer system 200 further includes a hash digest 280 of cryptographic hash values that identify the contents of one or more operating system components that have been loaded into regions in memory 270. It is noted that a cryptographic hash value is known in the art as being generated by a one-way function, mathematical or otherwise, which takes a variable-length input string, called a pre-image and converts it to a fixed-length, generally smaller, output string referred to as a hash value. The hash function is one-way in that it is difficult to generate a pre-image that matches the hash value of another pre-image. A hash digest signing engine 290 has a secure channel to access the hash digest 280 and will sign the contents of the hash digest 280 upon receiving a request to do so. Signing the contents of a hash digest 280 is known in the art, and is used to produce a digital signature that can be later used to authenticate the identity of the signer and to ensure that the content of the hash digest 280 has not been tampered with. By requesting such a signing, an outside entity may observe the state of system components reported by the hash and decide whether or not to trust

the computer system 200, i.e. whether or not the signed contents of the hash digest 280 match the expected signature of the system components.

[0014] In order to insure that the state of the components reported by the hash are such that the computer system 200 can be trusted, each of the computer system's CPUs 210/220/230 incorporate or is capable of incorporating an embodiment of the method and apparatus of the present invention to facilitate the installation (or loading) of a trustable operating system.

[0015] In one embodiment, the method and apparatus of the present invention include a start secure operation (SSO) 206 and a join secure operation (JSO) 204, each of which are capable of operating on any of the computer system's CPUs 210/220/230. The SSO 206 and JSO 204 are logical operations that are performed atomically to insure the integrity of the computer system 200. The SSO 206 and JSO 204 may be implemented as a series of privileged instructions carried out in software, hardware, or a combination thereof without departing from the scope of the invention.

[0016] In one embodiment, the SSO 206 takes a region (or regions) of memory 270 that was specified in a memory region parameter 202 and causes the computer system 200 to perform a number of operations that enable one of the CPUs 210/220/230 to load and register one or more components of operating system code in the specified region of memory 270 while the JSO 204 prevents the other CPUs from interfering. Upon the loading of the one or more operating system components, the JSO 204 and SSO 206 further force the CPUs 210/220/230 to jump to a known entry point in the now secured specified region of memory 270, also referred to as a security kernel 275, in a known, privileged state, i.e. a known state that allows access to the computer system's 200 resources in accordance with the CPU's corresponding high privilege level 170.

[0017] In one embodiment, once the region or regions in memory 270 to be secured is identified, via memory region parameter 202 or otherwise, the SSO 206 places the code that is to be secured into the identified region in memory 270, i.e. places the operating system code (or a portion thereof) into the security kernel 275. The code may be any code that is desired to be trusted, such as the privileged software nucleus 125 of the operating system 120 or, in a system with VM 110, the VMM core 150, the VM monitor core code.

[0018] In one embodiment, once the code is placed in the security kernel 275, the SSO 206 securely launches the operating system by registering the identity of the operating system code, e.g. the privileged software nucleus 125 or the VMM core 150. The SSO 206 registers the identity of the code by computing and recording a hash digest 180 of the code, and cryptographically signing the hash digest 180 using the hash digest signing engine 290. Once registered, the operating system becomes a trustable operating system, capable of verification by an outside entity.

[0019] In a computer system 200 with more than one CPU, as illustrated in FIG. 2, the computer system 200 must also be capable of preventing CPUs 220/230, other than the CPU 210 executing the SSO 206, from interfering with the secure launch of the trustable operating system. Accordingly, each CPU 210/220/230 is further provided with a JSO 204. When an SSO 206 is initiated on CPU 210, the SSO 206 signals the other CPUs 220/230 to execute a JSO 204.

[0020] In one embodiment, the JSO 204 forces the respective CPUs 220/230 to enter a special halted state and to signal their entry into the halted state to the initiating SSO CPU 210. When the initiating SSO CPU 210 receives halted signals from all of the other CPUs 220/230, the SSO 206 commences loading a trustable operating system by placing the desired code in the security kernel 275 and registering it. Once the CPU 210

that initiated the SSO 206 completes loading the trustable operating system, i.e. when the identity of the code in the security kernel 275 has been registered, the SSO 206 forces the CPU 210 to jump to a known entry point in the security kernel 275, which now has a known privileged state as a result of the operation of the SSO 206. In addition, the SSO 206 signals the other CPUs 220/230 to exit their respective special halted states. Upon exiting the halted states, the JSO 204 forces the CPUs 220/230 to also jump to a known entry point in the security kernel 275.

[0021] In one embodiment, the memory region parameter 202 is specified as a range of addresses in memory 270, and includes one or more pairs of start and stop addresses. However, other ways of specifying which region or regions in memory 270 are to be secured may be employed without departing from the scope of the invention. For example, an alternate embodiment of the memory region parameter 202 may be specified as a starting address and region length.

[0022] Turning now to FIGS. 3-5, the particular methods of the invention are described in terms of computer software with reference to a series of flow diagrams. The methods to be performed by a computer constitute computer programs made up of computer-executable instructions. Describing the methods by reference to a flow diagram enables one skilled in the art to develop such programs including such instructions to carry out the methods on suitably configured computers (the processor of the computer executing the instructions from computer-accessible media). The computer-executable instructions may be written in a computer programming language or may be embodied in firmware logic, or in micro-engine code, or the like. If written in a programming language conforming to a recognized standard, such instructions can be executed on a variety of hardware platforms and for interface to a variety of operating systems. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming

languages may be used to implement the teachings of the invention as described herein. Furthermore, it is common in the art to speak of software, in one form or another (e.g., program, procedure, process, application, and the like), as taking an action or causing a result. Such expressions are merely a shorthand way of saying that execution of the software by a computer causes the processor of the computer to perform an action or produce a result.

[0023] FIG. 3 is a flow diagram illustrating certain aspects of a method to be performed by a computing device executing one embodiment of the illustrated invention shown in FIG. 2. In particular, FIG. 3 illustrates some of the acts to be performed by a computer executing an SSO 206 that incorporates one embodiment of the invention. Processing begins at process 305, where one of the CPUs of computer system 200, for example CPU 210, prepares for executing an SSO 206 by insuring at process 310 that all of the other CPUs 220/230 of computer system 200 have performed a JSO 204. The JSO 204 causes the other CPUs 220/230 of computer system 200 to enter a halted state so that they cannot interfere with the SSO 206 and CPU 210 during the loading of the trustable operating system. In one embodiment, after all of the other CPUs 220/230 have been halted, the SSO 206 continues at process 315 to cause the CPU 210, or in some cases, the memory controller 260, to block the devices 240/245/250 of computer system 200 from accessing regions in memory 270 specified in memory region parameters 202, i.e. the security kernel 275. Blocking devices from accessing the security kernel 275 for the duration of the SSO 206 is typically only necessary in a computer system 200 that supports direct memory access (DMA). In one embodiment, blocking devices from accessing the security kernel 275 may also be performed by a standard chipset.

[0024] In one embodiment, at process 320, the SSO 206 erases the current contents of the hash digest 280 in preparation for recording current platform and hash digest information. At process 325, the SSO 206 records the platform information in the

hash digest 280. The recording of platform information may or may not be necessary, depending on the architecture of the computer system 200, and can include the version number of the CPU 210 executing the SSO 206, and the like. At process 330, the SSO 206 further computes a cryptographic hash digest of the code now present in the security kernel 275, i.e. the privileged software nucleus 125 or VMM core 150. The SSO 206 further records the information, also in the hash digest 280. At process 335, upon recording the necessary information in the hash digest 280, the SSO 206 places the CPU 210 into a known privileged state. Once the CPU 210 is in the known privileged state, the SSO 206 can further force the CPU 210 to jump to a known entry point in the security kernel 275. The known entry point may be any addressable area of the security kernel 275. Once the CPU 210 has jumped to the known entry point, the SSO 206 is complete and signals the other CPUs 220/230 to resume activity and returns control to the CPU 210.

[0025] Upon completion of SSO 206, an outside entity may send a request to the hash digest signing engine 290 to activate a secure channel to access the hash digest 280 and cause the digest signing engine 290 to read and cryptographically sign the content of the digest 280 recorded by the SSO 206. As noted earlier, by requesting such a signing, the outside entity may observe the state of components reported by the hash and decide whether or not to trust the computer system 200, i.e. whether or not a trustable operating system has been loaded.

[0026] FIG. 4 is a flow diagram illustrating certain aspects of a method to be performed by a computing device executing one embodiment of the illustrated invention shown in FIG. 2. In particular, FIG. 4 illustrates some of the acts to be performed by a computer executing a JSO 204 that incorporates one embodiment of the invention. Processing begins at process 405, where each of the computer system's 200 non-SSO CPUs, for example CPUs 220/230, enter a special halted state in response to the actions

of the SSO 206 on CPU 210. The halted state prevents the CPUs 220/230 from interfering with the SSO 206 and CPU 210 during the loading of the trustable operating system. The CPUs 220/230 each signal the SSO 206 on CPU 210 as they enter the halted state. The JSO 204 continues at decision process 415, which waits until receiving a signal that the SSO 206 on CPU 210 has completed the initialization of a trustable operating system. Once the initialization is complete the JSO 204 continues at process 420 causing the CPUs 220/230 to exit the special halted state. At process 425, the JSO 204 causes the CPUs 220/230 to jump to a known entry point in the security kernel 275, after which the JSO 204 completes processing at termination 430 and returns control to the respective CPUs 220/230.

[0027] While FIGS. 3-4 describe a generalized embodiment of the SSO 206 and JSO 204 processes, FIG. 5 describes an example implementation of the SSO 206 and JSO 204 on a computer system 200 with VM 110, including VM systems with 32-bit CPUs and VMM extensions 160. Processing begins at process 505, where the SSO 206 on one of the computer system's 200 CPUs, say CPU 210, receives memory region parameters 202 in the form of a start physical address, denoted as parameter EAX, and an end physical address, denoted as parameter ECX. Taken together, the addresses specified in the EAX and ECX parameters specify the region in memory 270 that is to be secured. The SSO 206 takes preparatory actions at process 510 to provide the required environment within which the SSO 206 will operate. The preparatory actions depend upon the architecture of the computer system 200 and may include, but are not limited to, insuring that the starting physical address, EAX, has a value that is less than the ending physical address, ECX. In addition, the SSO 206 can insure that the protected mode of the CPU 210 is enabled while the paging, physical address extension and VM extension modes are disabled, and that the privilege level of the CPU 210 is temporarily set to zero. Other possible preparatory actions might include disabling direct memory access (DMA)

to the region or regions in memory 270 that is or are to be secured, i.e. the security kernel 275, and disabling hardware interruptions to the CPU 210. Disabling hardware interruptions helps to insure that the SSO 206 and JSO 204 are performed atomically. Most importantly, the SSO 206 provides the required environment for loading a trustable operating system by causing each of the other CPUs 220/230 to commence a JSO 204 in order to insure that all of the non-SSO CPUs are halted, and thereby prevented from interfering with the operation of the SSO 206.

[0028] Upon completion of the preparatory actions, the SSO 206 continues at process 515 to create a cryptographic hash 280 for the specified region in memory 270 starting at address EAX and ending at address ECX. When securing multiple regions in memory 270, process 515 is repeated until all secured regions, i.e. the entire security kernel 275 are included in the cryptographic hash 280. At process 520, the SSO 206 records the cryptographic hash 280 in a chipset register that functions as the hash digest 280. The SSO 206 continues at process 525 by inducing the CPU 210 to enter a known state, and further at process 530 by causing the CPU 210 to jump to the hashed (i.e. the secured) region in memory 270, i.e. the security kernel 275. The SSO 206 concludes at process 535, where the CPU 210 will be in the known induced state with all interruptions disabled, and the security kernel 275 will be secured.

[0029] FIG. 6 illustrates one embodiment of an general purpose computer system 600 in which one embodiment of the invention illustrated in FIGS. 2-5 may be practiced. One embodiment of the present invention may be implemented on a personal computer (PC) architecture. However, it will be apparent to those of ordinary skill in the art that alternative computer system architectures or other processor, programmable or electronic-based devices may also be employed.

[0030] In general, such computer systems as illustrated in FIG. 6 include one or more processors 602 coupled through a bus 601 to a random access memory (RAM) 603, a read only memory (ROM) 604, and a mass storage device 607. Mass storage device 607 represents a persistent data storage device, such as a floppy disk drive, fixed disk drive (e.g., magnetic, optical, magneto-optical, or the like), or streaming tape drive. Processor 602 represents a central processing unit of any type of architecture, such as complex instruction set computer (CISC), reduced instruction set computer (RISC), very long instruction word (VLIW), or hybrid architecture. In one embodiment the processors 602 are compatible with an Intel Architecture (IA) processor, such as the Pentium ™ series, the IA-32 ™ and the IA-64 ™. In one embodiment, the computer system 600 includes any number of processors such as the CPUs 210/220/230 illustrated in FIG. 2.

[0031] Display device 605 is coupled to processor(s) 602 through bus 601 and provides graphical output for computer system 600. Input devices 606 such as a keyboard or mouse are coupled to bus 601 for communicating information and command selections to processor 602. Also coupled to processor 602 through bus 601 is an input/output interface 610 which can be used to control and transfer data to electronic devices (printers, other computers, etc.) connected to computer system 600. Computer system 600 includes network devices 608 for connecting computer system 600 to a network 614 through which data may be received, e.g., from remote device 612. Network devices 608, may include Ethernet devices, phone jacks and satellite links. It will be apparent to one of ordinary skill in the art that other network devices may also be utilized.

[0032] One embodiment of the invention may be stored entirely as a software product on mass storage 607. Another embodiment of the invention may be embedded in a hardware product, for example, in a printed circuit board, in a special purpose processor, or in a specifically programmed logic device communicatively coupled to bus

601. Still other embodiments of the invention may be implemented partially as a software product and partially as a hardware product.

[0033] When embodiments of the invention are represented as a software product stored on a machine-accessible medium (also referred to as a computer-accessible medium or a processor-accessible medium) such as mass storage device 607, the machine-accessible medium may be any type of magnetic, optical, or electrical storage medium including a diskette, CD-ROM, memory device (volatile or non-volatile), or similar storage mechanism. The machine-accessible medium may contain various sets of instructions, code sequences, configuration information, or other data. Those of ordinary skill in the art will appreciate that other instructions and operations necessary to implement the described invention may also be stored on the machine-accessible medium. In one embodiment of the present invention, the machine-accessible medium includes instructions that when executed by a machine causes the machine to perform operations comprising the SSO 206 and JSO 204.

[0034] Accordingly, a novel method is described for loading a trustable operating system. From the foregoing description, those skilled in the art will recognize that many other variations of the present invention are possible. For example, when implementing the invention on a mainframe or comparable class of machine, it may not be necessary to disable direct memory access (DMA) to the region or regions in memory 270 that is or are to be secured, i.e. the security kernel 275, or to disable hardware interruptions to the CPU 210. On the other hand, when implementing the invention on a PC-architected machine, such additional protective mechanisms may be needed to provide an operating environment within which the invention may be practiced. Thus, the present invention is not limited by the details described. Instead, the present invention can be practiced with modifications and alterations within the spirit and scope of the appended claims.